> Note: I have only mentioned questions that are must-to-know

## Theory Javascript Questions:

**1. What are the different data types present in javascript?**
**2. Explain Hoisting in JavaScript?**

Hoisting works differently for arrow functions and normal functions. Normal functions are fully hoisted, meaning both their name and definition are moved to the top of the scope. In contrast, arrow functions are not fully hoisted because they are assigned to variables. Only the variable is hoisted, not the arrow function itself.

**3. Difference between `var`, `let`, and `const`keyword in javascript?**
**4. What is pass-by-value and pass-by-reference?**

All primitive data types are passed by values(A copy of the value is used) and all non-primitive data types are passed by referenced (The reference to the object is passed by value).

**5. What is the difference between a deep copy and a shallow copy?**
**6. What are Self-Invoking Functions, or Immediately Invoked Function Expressions (IIFE), also known as Anonymous Functions?**

```
(function () {
    console.log("I am a self-invoking function!");
})();
```

**7. What do you mean by "strict mode" in JavaScript?**
**8. Explain Higher-Order Functions in JavaScript.**
**9. Explain the `this` keyword in JavaScript.**
**10. Explain the `call()`, `apply()`, and `bind()` methods in JavaScript.**
**11. What is currying in JavaScript?**
**12. What is lexical scoping in JavaScript?**
**13. Explain Closures in JavaScript.**

### 14. What are object prototypes?
### 15. What is prototypal inheritance?
### 16. What are callbacks in JavaScript?
### 17. Explain callback hell.
### 18. What is the rest parameter, and how does it differ from the spread operator?
### 19. What is the use of Promises in JavaScript?
### 20. What are generator functions?
### 21. What is a Temporal Dead Zone in JavaScript?
### 22. What are `async` and `await` in JavaScript?
### 23. Explain the `reduce()` function in JavaScript
### 24. What is implicit coercion in JavaScript?
### 25. What does it mean for functions to be "first-class citizens" in JavaScript?
### 26. Explain the scope of `this` inside an object.

If a function is inside an object, `this` refers to the object and can access its properties.

If `this` is used inside a nested function or object, it will refer to the outermost object, which is usually the global window object.

### 27. What is the significance of the `new` keyword in JavaScript?
### 28. What is memoization in JavaScript?
### 29. What are `Map`, `WeakMap`, and `WeakSet` in JavaScript?
### 30. What is event propagation in JavaScript?
### 31. What is event delegation in JavaScript?
### 32. What is the event loop in JavaScript?
### 33. What is control flow in JavaScript?

Control flow in JavaScript determines the order in which statements are executed in your code. Here's a breakdown of the main control flow structures: If Statement, Using If-Else Statement, Using Switch Statement, Using the Ternary Operator, Using For loop.

### 34. What is the difference between Server-Side Rendering (SSR) and Client-Side Rendering (CSR)?
### 35. What is the difference between declarative and imperative programming?

Declarative Programming: Describes what to do, focusing on the desired outcome rather than the steps to achieve it.

Imperative Programming: Describes how to do it, specifying the exact steps to achieve the desired result.

### 36. What is debouncing and throttling?

## Feature Based Questions:

### 1. Reverse a string in javascript?

```
const reversestring = str => str.split("").reverse().join("");
const res = reversestring("tpircsavaJ iH")
console.log(res) // Hi Javascript
```

### 2. Write code for debouncing.

```
<input placeholder="Enter test" id="userInput"/>
```

```
const inputBox = document.getElementById('userInput');

inputBox.addEventListener('input', (event) => {
  Debouncing(event);
});

let interval;

function Debouncing (event)  {
clearTimeout(interval)
interval = setTimeout(() => {
      console.log(event.target.value);
}, 2000)
}
```

### 3. Write code for throttling.

```
<input placeholder="Enter test" id="userInput"/>
```

```
const inputBox = document.getElementById('userInput');

inputBox.addEventListener('input', (event) => {
  Throttling(event);
});

let value = true;

function Throttling (event)  {
  if (value === true) {
    console.log(event.target.value);
    value = false;
    setTimeout(() => {
        value = true;
    }, 2000)
  }
}
```

### 4. How to sort the array of objects based on a key?

```
const array = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 20 }
];

// Sort by age (ascending)
array.sort((a, b) => a.age - b.age);

console.log(array);
// Output:
// [
//   { name: 'Charlie', age: 20 },
//   { name: 'Alice', age: 25 },
//   { name: 'Bob', age: 30 }
// ]
```

### 5. write a constructor function.

```
function Person(name, age, city) {
  // Properties
  this.name = name;
  this.age = age;
  this.city = city;

  // Method
  this.greet = function () {
    console.log(`Hi, I'm ${this.name}, and I live in ${this.city}.`);
  };
}

// Creating an instance
const person1 = new Person("Prajesh", 25, "Pune");
const person2 = new Person("Anjali", 28, "Mumbai");

// Using the method
person1.greet(); // Output: Hi, I'm Prajesh, and I live in Pune.
person2.greet(); // Output: Hi, I'm Anjali, and I live in Mumbai.
```

### 6. Write the polyfill of a map, reduce, and filter.

Here A polyfill is a piece of code that mimics the functionality or feature of a newer API or method, enabling it to work in environments where it is not natively supported.

Map

```
Array.prototype.myMap = function(callback) {
  if (!this || !Array.isArray(this)) {
    throw new TypeError('myMap can only be called on arrays.');
  }

  const result = [];
  for (let i = 0; i < this.length; i++) {
    if (this.hasOwnProperty(i)) {
      result.push(callback(this[i], i, this));
    }
  }
  return result;
};

// Example usage
const numbers = [1, 2, 3];
const doubled = numbers.myMap((num) => num * 2);
console.log(doubled); // Output: [2, 4, 6]
```

### Reduce

```
Array.prototype.myReduce = function(callback, initialValue) {
  if (!this || !Array.isArray(this)) {
    throw new TypeError('myReduce can only be called on arrays.');
  }
  if (typeof callback !== 'function') {
    throw new TypeError('Callback must be a function.');
  }

  let accumulator = initialValue;
  let startIndex = 0;

  if (accumulator === undefined) {
    if (this.length === 0) {
      throw new TypeError('Reduce of empty array with no initial value.');
    }
    accumulator = this[0];
    startIndex = 1;
  }

  for (let i = startIndex; i < this.length; i++) {
    if (this.hasOwnProperty(i)) {
      accumulator = callback(accumulator, this[i], i, this);
    }
  }

  return accumulator;
};

// Example usage
const sum = [1, 2, 3, 4].myReduce((acc, num) => acc + num, 0);
console.log(sum); // Output: 10
```

Filter

```
Array.prototype.myFilter = function(callback) {
  if (!this || !Array.isArray(this)) {
    throw new TypeError('myFilter can only be called on arrays.');
  }
  if (typeof callback !== 'function') {
    throw new TypeError('Callback must be a function.');
  }

  const result = [];
  for (let i = 0; i < this.length; i++) {
    if (this.hasOwnProperty(i) && callback(this[i], i, this)) {
      result.push(this[i]);
    }
  }
  return result;
};

// Example usage
const numbers = [1, 2, 3, 4];
const evens = numbers.myFilter((num) => num % 2 === 0);
console.log(evens); // Output: [2, 4]
```

**7. Write three functions, each using setTimeout to delay execution for 3 seconds, 2 seconds, and 1 second respectively. Then, write a function to execute these three functions in the same order (one, two, three) such that the output is:**

```
one
two
three
```

```
const one = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
    console.log("one")
    resolve();
    }, 3000)
  })
}

const two = () => {
     return new Promise((resolve, reject) => {
    setTimeout(() => {
    console.log("two")
    resolve();
    }, 2000)
  })
}

const three = () => {
     return new Promise((resolve, reject) => {
    setTimeout(() => {
    console.log("three")
    resolve();
    }, 1000)
  })
}

const check = async () => {
await one();
await two();
await three();
}

check();

// output:
// one
// two
// three
```

## 8. Write a simple callback function?

```
// A function that takes a callback as an argument
function greetUser(name, callback) {
    console.log(`Hello, ${name}!`);
    callback();
}

// A callback function
function displayMessage() {
    console.log('Welcome to the platform!');
}

// Using the greetUser function with the callback
greetUser('Jhon', displayMessage);
```

### 9. Write a multiply function which will properly work when invoked below syntax.

```
console.log(multiply (2) (3) (4)) // output: 24
console.log(multiply (3) (4) (5)) // output: 60
```

```
function multiply (x) {
  return function (y) {
    return function (z) {
      return x * y * z
    }
  }
}
```

### 10. Write code for memoization.

```
const memoizeAddition = () => {
  let cache = {};
  return (value) => {
    if (value in cache) {
      console.log("Fetching from cache");
      return cache[value]; // Here, cache.value cannot be used as property name
starts with the number which is not a valid JavaScript  identifier. Hence, can
only be accessed using the square bracket notation.
    } else {
      console.log("Calculating result");
      let result = value + 20;
      cache[value] = result;
      return result;
    }
  };
};
// returned function from memoizeAddition
const addition = memoizeAddition();
console.log(addition(20)); //output: 40 calculated
console.log(addition(20)); //output: 40 cached
```

Congratulations on making it to the bottom of this post, and thank you so much for reading!